

# Mathematical Programming Strategies for Solving the Minimum Common String Partition Problem

Christian Blum<sup>1,2</sup>, José A. Lozano<sup>1</sup>, and Pedro Pinacho Davidson<sup>1,3</sup>

<sup>1</sup>Department of Computer Science and Artificial Intelligence  
University of the Basque Country UPV/EHU, San Sebastian, Spain  
{christian.blum,ja.lozano}@ehu.es

<sup>2</sup>IKERBASQUE  
Basque Foundation for Science, Bilbao, Spain

<sup>3</sup>Escuela de Informática  
Universidad Santo Tomás, Concepción, Chile  
ppinacho@santotomas.cl

## Abstract

The minimum common string partition problem is an NP-hard combinatorial optimization problem with applications in computational biology. In this work we propose the first integer linear programming model for solving this problem. Moreover, on the basis of the integer linear programming model we develop a deterministic 2-phase heuristic which is applicable to larger problem instances. The results show that provenly optimal solutions can be obtained for problem instances of small and medium size from the literature by solving the proposed integer linear programming model with CPLEX. Furthermore, new best-known solutions are obtained for all considered problem instances from the literature. Concerning the heuristic, we were able to show that it outperforms heuristic competitors from the related literature.

## 1 Introduction

Optimization problems related to strings—such as protein or DNA sequences—are very common in bioinformatics. Examples include string consensus problems such as the far-from most string problem [20, 19], the longest common subsequence problem and its variants [14, 22], and alignment problems [12]. These problems are often computationally very hard, if not even *NP*-hard [9]. In this work we deal with the *minimum common string partition* (MCSP) problem, which can be described as follows. We are given two related input strings that have to be partitioned each into the same collection of substrings. The size of the collection is subject to minimization. A formal description of the problem will be provided in Section 1.1. The MCSP problem has applications, for example, in the bioinformatics field. Chen et al. [2] point out that the MCSP problem is closely related to the problem of sorting by reversals with duplicates, a key problem in genome rearrangement.

In this paper we introduce the first integer linear program (ILP) for solving the MCSP problem. An experimental evaluation on problem instances from the related literature shows that this ILP can be efficiently solved, for example, by using any version of IBM ILOG CPLEX. However, a study on new instances of larger size demonstrates the limitations of the model. Therefore, we additionally introduce a deterministic 2-phase heuristic which is strongly based on the original ILP. The experimental evaluation shows that the heuristic is applicable to larger problem instances than the original ILP. Moreover, it is shown that the heuristic outperforms competitor algorithms from the related literature on known problem instances.

## 1.1 Problem Description

The MCSP problem can technically be described as follows. Given are two input strings  $s_1$  and  $s_2$ , both of length  $n$  over a finite alphabet  $\Sigma$ . These two strings are required to be *related*, which means that each letter appears the same number of times in each of them. Note that this definition implies that  $s_1$  and  $s_2$  have the same length. A valid solution to the MCSP problem is obtained by partitioning  $s_1$  into a set  $P_1$  of non-overlapping substrings, and  $s_2$  into a set  $P_2$  of non-overlapping substrings, such that  $P_1 = P_2$ . Moreover, we are interested in finding a valid solution such that  $|P_1| = |P_2|$  is minimal.

Consider the following example. Given are DNA sequences  $s_1 = \mathbf{AGACTG}$  and  $s_2 = \mathbf{ACTAGG}$ . Obviously,  $s_1$  and  $s_2$  are related because **A** and **G** appear twice in both input strings, while **C** and **T** appear once. A trivial valid solution can be obtained by partitioning both strings into substrings of length 1, that is,  $P_1 = P_2 = \{\mathbf{A}, \mathbf{A}, \mathbf{C}, \mathbf{T}, \mathbf{G}, \mathbf{G}\}$ . The objective function value of this solution is 6. However, the optimal solution, with objective function value 3, is  $P_1 = P_2 = \{\mathbf{ACT}, \mathbf{AG}, \mathbf{G}\}$ .

## 1.2 Related Work

The MCSP problem has been introduced by Chen et al. [2] due to its relation to genome rearrangement. More specifically, it has applications in biological questions such as: May a given DNA string possibly be obtained by rearrangements of another DNA string? The general problem has been shown to be *NP*-hard even in very restrictive cases [10]. Other papers concerning problem hardness consider, for example, the  $k$ -MCSP problem, which is the version of the MCSP problem in which each letter occurs at most  $k$  times in each input string. The 2-MCSP problem was shown to be APX-hard in [10]. When the input strings are over an alphabet of size  $c$ , the corresponding problem is denoted as  $\text{MCSP}^c$ . Jiang et al. proved that the decision version of the  $\text{MCSP}^c$  problem is *NP*-complete when  $c \geq 2$  [15].

The MCSP has been considered quite extensively by researchers dealing with the approximability of the problem. Cormode and Muthukrishnan [4], for example, proposed an  $O(\log \log^* n)$ -approximation for the *edit distance with moves* problem, which is a more general case of the MCSP problem. Shapira and Storer [21] extended on this result. Other approximation approaches for the MCSP problem have been proposed in [18]. In this context, Chrobak et al. [3] studied a simple greedy approach for the MCSP problem, showing that the approximation ratio concerning the 2-MCSP problem is 3, and for the 4-MCSP problem the approximation ratio is  $\Omega(\log(n))$ . In the case of the general MCSP problem, the approximation ratio is between  $\Omega(n^{0.43})$  and  $O(n^{0.67})$ , assuming that the input strings use an alphabet of size  $O(\log(n))$ . Later Kaplan and Shafir [16] raised the lower bound to  $\Omega(n^{0.46})$ .

Kolman proposed a modified version of the simple greedy algorithm with an approximation ratio of  $O(k^2)$  for the  $k$ -MCSP [17]. Recently, Goldstein and Lewenstein proposed a greedy algorithm for the MCSP problem that runs in  $O(n)$  time (see [11]). He [13] introduced a greedy algorithm with the aim of obtaining better average results.

Damaschke [5] was the first one to study the fixed-parameter tractability (FPT) of the problem. Later, Jiang et al. [15] showed that both the  $k$ -MCSP and  $\text{MCSP}^c$  problems admit FPT algorithms when  $k$  and  $c$  are constant parameters. Finally, Fu et al. [8] proposed a  $O(2^n n^{O(1)})$  time algorithm for the general case and an  $O(n(\log n)^2)$  time algorithm applicable under some constraints.

To our knowledge, the only metaheuristic approaches that have been proposed in the related literature for the MCSP problem are (1) the  $\mathcal{MA\mathcal{X}}\text{-}\mathcal{MIN}$  Ant System by Ferdous and Sohel [6, 7] and (2) the probabilistic tree search algorithm by Blum et al. [1]. Both works applied their algorithm to a range of artificial and real DNA instances from [6].

### 1.3 Organization of the Paper

The remaining part of the paper is organized as follows. In Section 2, the ILP model for solving the MCSP is outlined. Moreover, an experimental evaluation is provided. The deterministic heuristic, together with an experimental evaluation, is described in Section 3. Finally, in Section 4 we provide conclusions and an outlook to future work.

## 2 An Integer Linear Program to Solve the MCSP

In the following we present the first ILP model for solving the MCSP. For this, the definitions provided in the following are required. Note that an illustrative example is provided in Section 2.3.

### 2.1 Preliminaries

Henceforth, a *common block*  $b_i$  of input strings  $s_1$  and  $s_2$  is denoted as a triple  $(t_i, k1_i, k2_i)$  where  $t_i$  is a string which can be found starting at position  $1 \leq k1_i \leq n$  in string  $s_1$  and starting at position  $1 \leq k2_i \leq n$  in string  $s_2$ . Moreover, let  $B = \{b_1, \dots, b_m\}$  be the (ordered) set of all possible common blocks of  $s_1$  and  $s_2$ .<sup>1</sup> Given the definition of  $B$ , any valid solution  $\mathcal{S}$  to the MCSP problem is a subset of  $B$ —that is,  $\mathcal{S} \subset B$ —such that:

1.  $\sum_{b_i \in \mathcal{S}} |t_i| = n$ , that is, the sum of the length of the strings corresponding to the common blocks in  $\mathcal{S}$  is equal to the length of the input strings.
2. For any two common blocks  $b_i, b_j \in \mathcal{S}$  it holds that their corresponding strings neither overlap in  $s_1$  nor in  $s_2$ .

Moreover, any (valid) partial solution  $\mathcal{S}_{\text{partial}}$  is a subset of  $B$  fulfilling the following conditions: (1)  $\sum_{b_i \in \mathcal{S}_{\text{partial}}} |t_i| < n$  and (2) for any two common blocks  $b_i, b_j \in \mathcal{S}_{\text{partial}}$  it holds that their corresponding strings neither overlap in  $s_1$  nor in  $s_2$ . Note that any valid partial solution can be extended to be a valid solution. Furthermore, given a partial solution  $\mathcal{S}_{\text{partial}}$ , set  $B(\mathcal{S}_{\text{partial}}) \subset B$  denotes the set of common blocks that may be used in order to extend  $\mathcal{S}_{\text{partial}}$  such that the result is again a valid (partial) solution.

---

<sup>1</sup>The way in which  $B$  is ordered is of no importance.

## 2.2 The Integer Linear Program

First, two binary  $m \times n$  matrices  $M1$  and  $M2$  are defined as follows. In both matrices, row  $1 \leq i \leq m$  corresponds to common block  $b_i \in B$ . Moreover, a column  $1 \leq j \leq n$  corresponds to position  $j$  in input string  $s_1$ , respectively  $s_2$ . In general, the entries of matrix  $M1$  are set to zero. However, in each row  $i$ , the positions that string  $t_i$  (of common block  $b_i$ ) occupies in input string  $s_1$  are set to one. Correspondingly, the entries of matrix  $M2$  are set to zero, apart from the fact that in each row  $i$  the positions occupied by string  $t_i$  in input string  $s_2$  are set to one. Henceforth, the position  $(i, j)$  of a matrix  $M$  is denoted by  $M_{i,j}$ . Finally, we introduce for each common block  $b_i \in B$  a binary variable  $x_i$ . With these definitions we can express the MCSP in form of the following integer linear program, henceforth referred to by  $\text{ILP}_{\text{orig}}$ .

$$\begin{aligned}
 & \min \sum_{i=1}^m x_i & (1) \\
 & \text{subject to:} \\
 & \sum_{i=1}^m |t_i| \cdot x_i = n & (2) \\
 & \sum_{i=1}^m M1_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \dots, n & (3) \\
 & \sum_{i=1}^m M2_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \dots, n & (4) \\
 & x_i \in \{0, 1\} \quad \text{for } i = 1, \dots, m
 \end{aligned}$$

Hereby, the objective function minimizes the number of selected common blocks. Constraint (2) ensures that the sum of the length of the strings corresponding to the selected common blocks is equal to  $n$ . Finally, constraints (3) make sure that the strings corresponding to the selected common blocks do not overlap in input string  $s_1$ , while constraints (4) make sure that the strings corresponding to the selected common blocks do not overlap in input string  $s_2$ .

## 2.3 Example

As an example, consider the small problem instance from Section 1.1. The complete set of common blocks ( $B$ ) as induced by input strings  $s_1 = \mathbf{AGACTG}$  and  $s_2 = \mathbf{ACTAGG}$  is as

follows:

$$B = \left\{ \begin{array}{l} b_1 = (\mathbf{ACT}, 3, 1) \\ b_2 = (\mathbf{AG}, 1, 4) \\ b_3 = (\mathbf{AC}, 3, 1) \\ b_4 = (\mathbf{CT}, 4, 2) \\ b_5 = (\mathbf{A}, 1, 1) \\ b_6 = (\mathbf{A}, 1, 4) \\ b_7 = (\mathbf{A}, 3, 1) \\ b_8 = (\mathbf{A}, 3, 4) \\ b_9 = (\mathbf{C}, 4, 2) \\ b_{10} = (\mathbf{T}, 5, 3) \\ b_{11} = (\mathbf{G}, 2, 5) \\ b_{12} = (\mathbf{G}, 2, 6) \\ b_{13} = (\mathbf{G}, 6, 5) \\ b_{14} = (\mathbf{G}, 6, 6) \end{array} \right\}$$

Given set  $B$ , matrices  $M1$  and  $M2$  are the following ones:

$$M1 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad M2 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The optimal solution to this instance is  $\mathcal{S} = \{b_1, b_2, b_{14}\}$ . It can easily be verified that this solution respects constraints (2-4) of the ILP model.

## 2.4 Experimental Evaluation

In the following we will provide an experimental evaluation of model  $ILP_{orig}$ . The model was implemented in ANSI C++ using GCC 4.7.3 for compiling the software. Moreover, the model was solved with IBM ILOG CPLEX V12.1. The experimental results that we outline in the following were obtained on a cluster of PCs with "Intel(R) Xeon(R) CPU 5130" CPUs of 4 nuclei of 2000 MHz and 4 Gigabyte of RAM.

### 2.4.1 Problem Instances

For testing model  $ILP_{orig}$  we chose the same set of benchmark instances that was used by Ferdous and Sohel in [6] for the experimental evaluation of their ant colony optimization approach. This set contains, in total, 30 artificial instances and 15 real-life instances consisting

of DNA sequences. Remember, in this context, that each problem instance consists of two related input strings. Moreover, the benchmark set consists of four subsets of instances. The first subset (henceforth labelled GROUP1) consists of 10 artificial instances in which the input strings are maximally of length 200. The second set (GROUP2) consists of 10 artificial instances with input string lengths between 201 and 400. In the third set (GROUP3) the input strings of the 10 artificial instances have lengths between 401 and 600. Finally, the fourth set (REAL) consists of 15 real-life instances of various lengths.

## 2.4.2 Results

The results are shown in Tables 1-4, in terms of one table per instance set. The structure of these tables is as follows. The first column provides the instance identifiers. The second column contains the results of the greedy algorithm from [3] (results were taken from [6]). The third column provides the value of the best solution found in four independent runs per problem instance (with a CPU time limit of 7200 seconds per run) by the ACO approach by Ferdous and Sohel [6, 7].<sup>2</sup> The fourth column provides the value of the best solution found in 10 independent runs per problem instance (with a CPU time limit of 1000 seconds per run) by the probabilistic tree search algorithm (henceforth labelled TRESEA) by Blum et al. [1]. TRESEA was run on the same machines as the ones used for the current work. Finally, the last four table columns are dedicated to the presentation of the results provided by solving model  $ILP_{orig}$ . The first one of these columns provides the value of the best solution found within 3600 CPU seconds. In case the optimality of the corresponding solution was proved by CPLEX, the value is marked by an asterisk. The second column dedicated to  $ILP_{orig}$  provides the computation time (in seconds). In case of having solved the corresponding problem to optimality, this column only displays one value indicating the time needed by CPLEX to solve the problem. Otherwise, this column provides two values in the form X/Y, where X corresponds to the time at which CPLEX was able to find the first valid solution, and Y corresponds to the time at which CPLEX found the best solution within 3600 CPU seconds. The third one of the columns dedicated to  $ILP_{orig}$  shows the optimality gap, which refers to the gap between the value of the best valid solution and the current lower bound at the time of stopping a run. Finally, the last column indicates the size of set  $B$ , that is, the size of the complete set of common blocks. Note that this value corresponds to the number of variables used by  $ILP_{orig}$ . The best result (among all algorithms) for each problem instance is marked by a grey background, and the last row of each table provides averages over the whole table.

The following conclusions can be drawn when analyzing the results. First, CPLEX is able to solve all instances of GROUP1 to optimality. This is done, on average, in about 13 seconds. Moreover, none of the existing algorithms was able to find any of these optimal solutions. Second, CPLEX was also able to find new best-known solutions for all remaining 35 problem instances, even though it was not able to prove optimality within 3600 CPU seconds, which is indicated by the positive optimality gaps. An exception is instance 1 of set REAL which also could be solved to optimality. Third, the improvements over the competitor algorithms obtained by solving  $ILP_{orig}$  with CPLEX are remarkable. In particular, the average improvement (in percent) over TRESEA, the best competitor from the literature, is 4.8% in the case of GROUP1, 9.2% in the case of GROUP2, 9.7% in the case of GROUP3, and 9.8% in the case

---

<sup>2</sup>In this context, note that the experiments for ACO were performed on a computer with an "Intel(R) 2 Quad" CPU with 2.33 GHz and 4 GB of RAM.

Table 1: Results for the 10 instances of GROUP1.

id	GREEDY	ACO	TRESEA	ILP <sub>orig</sub>			
	value	best	best	value	time (s)	gap	B
1	46	42	42	*41	1	0.0%	4299
2	56	51	48	*47	3	0.0%	6211
3	62	55	56	*52	30	0.0%	8439
4	46	43	43	*41	2	0.0%	4299
5	44	43	41	*40	1	0.0%	4718
6	48	42	41	*40	3	0.0%	4435
7	65	60	60	*55	38	0.0%	8687
8	51	47	45	*43	3	0.0%	4995
9	46	45	43	*42	2	0.0%	4995
10	63	59	58	*54	50	0.0%	9699
avg.	52.7	48.7	47.7	45.5	13.3	0.0%	6029.3

Table 2: Results for the 10 instances of GROUP2.

id	GREEDY	ACO	TRESEA	ILP <sub>orig</sub>			
	value	best	best	value	time (s)	gap	B
1	119	113	111	98	66/1969	2.9%	37743
2	122	118	114	106	129/1032	7.5%	47174
3	114	111	107	97	55/1216	2.7%	36979
4	116	115	111	102	63/949	4.9%	40960
5	135	132	127	116	146/3299	6.7%	52697
6	108	105	102	93	56/1419	5.6%	35650
7	108	98	96	88	41/2776	6.0%	30839
8	123	118	114	104	101/2980	5.1%	42668
9	124	119	113	104	81/1630	5.2%	42998
10	105	101	98	89	32/1458	3.6%	31169
avg.	117.4	113.0	109.3	99.7	77/1873	5.0%	39887.7

Table 3: Results for the 10 instances of GROUP3.

id	GREEDY	ACO	TRESEA	ILP <sub>orig</sub>			
	value	best	best	value	time (s)	gap	B
1	182	177	171	155	733/1398	7.5%	110973
2	175	175	168	155	553/869	7.7%	102670
3	196	187	185	166	746/2183	8.5%	119287
4	192	184	179	159	731/1200	6.9%	114975
5	176	171	163	150	485/886	9.7%	99775
6	170	160	162	147	399/764	9.1%	88839
7	173	167	161	149	524/990	9.8%	95765
8	185	175	169	151	492/3584	6.7%	97400
9	174	172	169	158	571/1186	10.9%	104186
10	171	167	161	148	547/1446	9.1%	98237
avg.	179.4	173.5	168.8	153.8	578/1451	8.6%	103211.0

of REAL.

In order to study the limits of solving ILP<sub>orig</sub> with CPLEX we randomly generated larger DNA instances. In particular, we generated one random instance for each input string size from {800, 1000, 1200, 1400, 1600, 1800, 2000}. CPLEX was stopped when at least 3600 CPU seconds had passed and at least one feasible solution had been found. However, if after 12 CPU hours still no feasible solution was found, the execution was stopped as well. The

Table 4: Results for the 15 instances of set REAL.

id	GREEDY	ACO	TRESEA	ILP <sub>orig</sub>			
	value	best	best	value	time (s)	gap	B
1	95	87	86	*78	972	0.0%	22799
2	161	155	154	139	432/752	9.2%	80523
3	121	116	113	104	125/3580	5.6%	45869
4	173	164	158	144	577/1730	6.5%	91663
5	172	171	165	150	778/2509	7.9%	108866
6	153	145	143	128	257/3578	6.5%	70655
7	140	140	131	121	359/2187	6.9%	73502
8	134	130	128	116	275/3365	6.8%	65560
9	149	146	142	131	399/613	8.8%	75833
10	151	148	144	131	311/1771	7.2%	69560
11	126	124	121	110	205/3711	4.8%	56160
12	143	137	138	126	299/793	9.8%	70861
13	180	180	171	156	784/1130	7.1%	115810
14	152	147	146	134	370/2456	9.7%	73449
15	157	160	152	139	560/1762	7.7%	91060
avg.	147.1	143.3	139.5	127.1	409/2131	7.0%	74163.9

Table 5: Results of applying CPLEX to ILP<sub>orig</sub> in the context of larger instances.

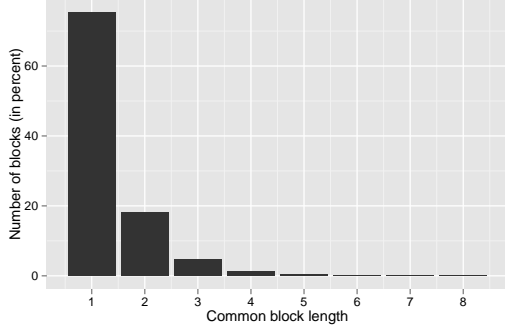
length	value	time (s)	gap	B
800	210	3228	10.7%	214622
1000	304	2922	26.4%	334411
1200	342	6220	22.6%	480908
1400	401	12124	24.9%	653401
1600	442	20616	24.1%	854500
1800	486	37304	24.0%	1084533
2000	n.a.	n.a.	n.a.	1335893

results are shown in Table 5. The first column of this table provides the length of the corresponding random instance. The remaining four columns contain the same information as already explained in the context of Tables 1-4, just that column **time (s)** simply provides the computation time (in seconds) at which the best solution was found. Analyzing the results we can observe that the application of CPLEX to ILP<sub>orig</sub> quickly becomes unpractical with growing input string size. For example, the first valid solution for the instance with string length 1400 was found after 20616 seconds. Concerning the largest problem instance, no valid solution was found within 12 CPU hours.

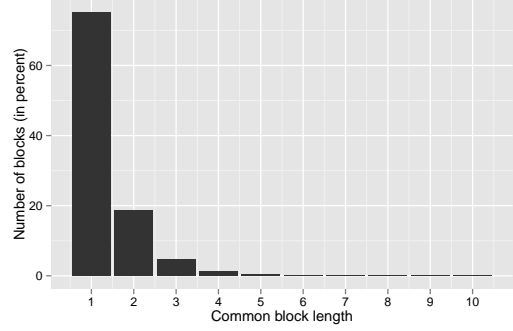
### 3 A MIP-Based Heuristic

As shown at the end of the previous section, the application of CPLEX to ILP<sub>orig</sub> reaches its limits starting from an input string size of about 1200. However, if it were possible to considerably reduce the size of the set of common blocks ( $B$ ), mathematical programming might still be an option to obtain good (heuristic) solutions. With this idea in mind we studied the distribution of the lengths of the strings of the common blocks in  $B$  for all 45 problem instances. This distribution is shown—averaged over the instances of each of the four instance sets—in Figure 1. Analyzing these distributions it can be observed, first of all,

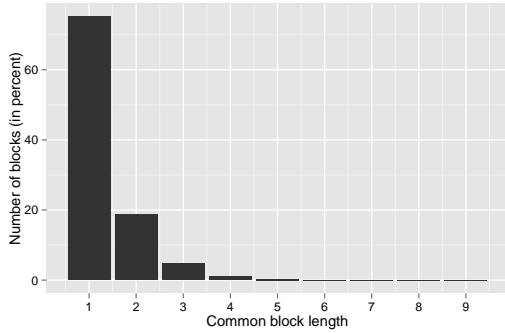




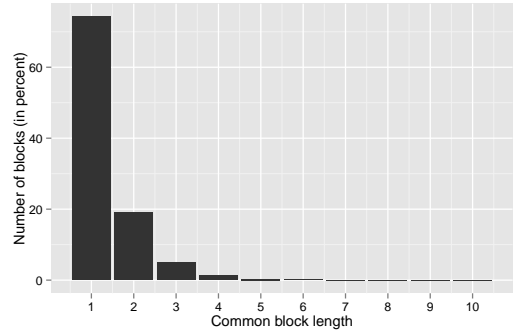
(a) Instances of set GROUP1.



(b) Instances of set GROUP2.



(c) Instances of set GROUP3.



(d) Instances of set REAL.

Figure 1: Distribution of the string lengths corresponding to the complete set of common blocks. The distributions are shown averaged over all instances for each of the four sets of problem instances.

that the distribution does not seem to depend on instance size.<sup>3</sup> However, the important aspect to observe is that around 75% of all the common blocks contain strings of length 1. Moreover, only a very small portion of these common blocks will form part of an optimal solution. In comparison, it is reasonable to assume that a much larger percentage of the blocks corresponding to large strings will form part of an optimal solution. These observations gave rise to the heuristic which is outlined in the following.

### 3.1 Heuristic

The proposed heuristic works in two phases. In the first phase, a subset of  $B$  (the complete set of common blocks) must be chosen. For this purpose, let  $B_{\geq l}$  (where  $l \geq 1$ ) denote the subset of  $B$  that contains all common blocks  $b_i$  from  $B$  with  $|t_i| \geq l$ , that is, all blocks whose corresponding string is longer or equal than  $l$ . Note, in this context, that  $B_{\geq 1} = B$ . Moreover, note that  $|B_{\geq 1}| \geq |B_{\geq 2}| \geq |B_{\geq 3}| \geq \dots \geq |B_{\geq \infty}|$ . Let  $l_{\max}$  be the smallest value for  $l$  such that  $|B_{\geq l_{\max}}| > 0$ . Observe that  $B_{\geq l_{\max}}$  only contains the common blocks with the longest strings. Having chosen a specific value for  $l$  from  $[2, l_{\max}]$ , the following ILP, henceforth referred to as  $\text{ILP}_{\text{ph1}}$ , may be solved.

<sup>3</sup>Most probably the distribution would change in some way when changing the size of the alphabet.

$$\begin{aligned}
& \max \sum_{b_i \in B_{\geq l}} (C \cdot |t_i| - 1) \cdot x_i & (5) \\
& \text{subject to:} \\
& \sum_{b_i \in B_{\geq l}} |t_i| \cdot x_i \leq n & (6) \\
& \sum_{b_i \in B_{\geq l}} M1_{i,j} \cdot x_i \leq 1 \quad \text{for } j = 1, \dots, n & (7) \\
& \sum_{b_i \in B_{\geq l}} M2_{i,j} \cdot x_i \leq 1 \quad \text{for } j = 1, \dots, n & (8) \\
& x_i \in \{0, 1\} \quad \text{for } b_i \in B_{\geq l}
\end{aligned}$$

ILP<sub>ph1</sub> is based on a binary variable  $x_i$  for each common block  $b_i \in B_{\geq l}$ . Moreover, matrices  $M1$  and  $M2$  are the same as the ones introduced in Section 2.2, that is, they are defined over the whole set  $B$ . The objective function basically maximizes the sum of the lengths of the strings corresponding to the chosen common blocks. However, the length of the string corresponding to each block is multiplied by a large-enough constant  $C$ , and the result is decremented by one. This has the following effect. In case of several solutions for which the sum of the lengths of the strings corresponding to the selected common blocks is equal, the program prefers the solution that reaches this sum with fewer common blocks. The constraints (6-8) are the same as in ILP<sub>orig</sub> (see Section 2.2), apart from the fact that all equality symbols are replaced by the  $\leq$ -symbol. In short, the idea of ILP<sub>ph1</sub> is to produce a partial solution for the original MCSP that covers as much as possible of both input strings, while choosing as few common blocks as possible.

Solving ILP<sub>ph1</sub> will henceforth be referred to as *phase 1* of the proposed heuristic. Let us denote by  $\mathcal{S}_{ph1}$  the solution provided by phase 1.<sup>4</sup> Due to the constraints of ILP<sub>ph1</sub> this solution is a valid partial solution to the original MCSP problem. The idea of the second phase is then to produce the best complete solution possible that contains  $\mathcal{S}_{ph1}$ . This is done by solving the following ILP, henceforth referred to as ILP<sub>ph2</sub>.

$$\begin{aligned}
& \min \sum_{b_i \in B_{ph2}} x_i & (9) \\
& \text{subject to:} \\
& \sum_{b_i \in B_{ph2}} |t_i| \cdot x_i = n & (10) \\
& \sum_{b_i \in B_{ph2}} M1_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \dots, n & (11) \\
& \sum_{b_i \in B_{ph2}} M2_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \dots, n & (12) \\
& x_i = 1 \quad \text{for } b_i \in \mathcal{S}_{ph1} & (13) \\
& x_i \in \{0, 1\} \quad \text{for } b_i \in B_{ph2}
\end{aligned}$$

<sup>4</sup>Remember that solutions are subsets of  $B$ .

Hereby,  $B_{ph2} := B(\mathcal{S}_{ph1}) \subset B$  is the set of common blocks that may be added to  $\mathcal{S}_{ph1}$  without violating any constraints.<sup>5</sup> Note that model  $ILP_{ph2}$  is the same as model  $ILP_{orig}$ , just that  $ILP_{ph2}$  only considers common blocks from  $B_{ph2}$  and that it forces any solution to contain all common blocks from  $\mathcal{S}_{ph1}$ ; see constraints (13). This completes the description of the heuristic.

### 3.2 Experimental Evaluation

Just like model  $ILP_{orig}$ , the heuristic was implemented in ANSI C++ using GCC 4.7.3 for compiling the software. The two ILP models were solved with IBM ILOG CPLEX V12.1, and the same machines as for the experimental evaluation of  $ILP_{orig}$  were used for running the experiments.

As mentioned before, the heuristic may be applied for any value of  $l$  from the interval  $[2, l_{max}]$ . In fact, we applied the heuristic to each of the 45 problem instances from sets GROUP1, GROUP2, GROUP3, and REAL, with all possible values for  $l$ . In order not to spend too much computation time the following stopping criterion was used for each call to CPLEX concerning any of the two involved ILP models. CPLEX was stopped (1) in case a provenly optimal solution was obtained or (2) in case at least 50 CPU seconds were spent and the first valid solution was obtained. The overall result of the heuristic for a specific problem instance is the value of the best solution found for any value of  $l$ . Moreover, as computation time we provide the sum of the computation times spend for all applications for different value of  $l$ .

The results are shown in Table 6, which contains one subtable for each of the four instance sets. Each subtable has the following format. The first column provides the instance identifier. The second column contains the value of the best solution found in the literature. Finally, the last two table columns present the results of our heuristic. The first one of these columns contains the value of the best solution generated by the heuristic, while the second column provides the total computation time (in seconds). The last row of each subtable presents averages over the whole subtable. Moreover, the best result for each instance is marked by a grey background, and those cases in which the result of applying CPLEX to  $ILP_{orig}$  could be matched are marked by a "+" symbol.

The results allow to make the following observations. First, our heuristic is able to improve the best-known result from the literature in 37 out of 46 cases. In further six cases the best-known results from the literature are matched. Finally, in two remaining cases the heuristic is not able to produce a solution that is at least as good as the best-known solution known from the literature. Overall, the heuristic improves by 3.4% (on average) over the best known results from the literature. On the downside, the heuristic is only able to match the results of applying CPLEX to model  $ILP_{orig}$  in three out of 45 cases. However, this changes with growing instance size, as we will show later in Section 3.4.

### 3.3 Gaining Insight into the Behavior of the Heuristic

With the aim of gaining more insight into the behavior of the heuristic with respect to the choice of a value for parameter  $l$ , the following information is presented in graphical form in Figure 2. Two graphics are shown for each of the four chosen problem instances. More precisely, we chose to present information for the largest problem instances from each of the four

---

<sup>5</sup>See Section 2.1 for the definition of  $B(\cdot)$ .

Table 6: Results of the heuristic. Each of the four subtables deals with one of the four problem instance sets.

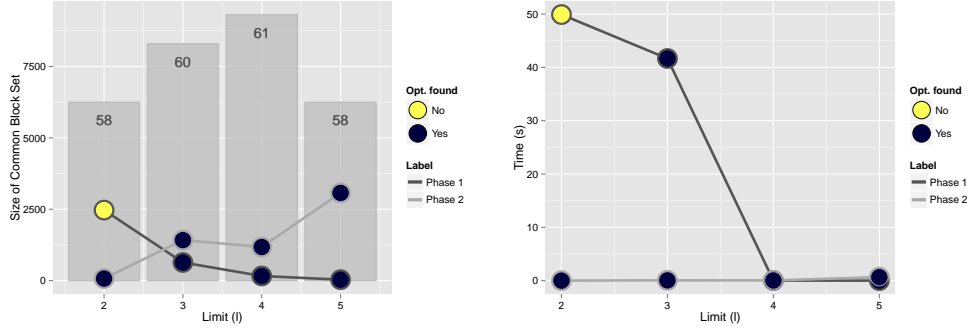
(a) Instances of GROUP1.				(b) Instances of GROUP2.			
id	BEST	Heuristic		id	BEST	Heuristic	
	KNOWN	value	time (s)		KNOWN	value	time (s)
1	42	42	7	1	111	103	207
2	48	48	30	2	114	110	214
3	55	53	60	3	107	99	299
4	43	43	1	4	111	105	261
5	41	+40	15	5	127	120	270
6	41	+40	6	6	102	97	252
7	60	57	22	7	96	91	166
8	45	44	1	8	114	108	223
9	43	46	6	9	113	109	160
10	58	58	92	10	98	94	158
avg.	47.7	47.1	24.0	avg.	109.3	103.6	221.0

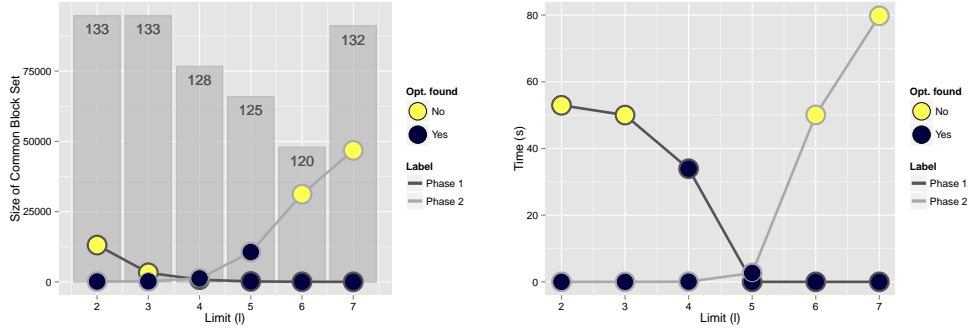
(c) Instances of GROUP3.				(d) Instances of REAL.			
id	BEST	Heuristic		id	BEST	Heuristic	
	KNOWN	value	time (s)		KNOWN	value	time (s)
1	171	172	1030	1	86	80	142
2	168	165	671	2	154	144	680
3	185	180	1186	3	113	112	232
4	179	171	970	4	158	157	667
5	163	163	610	5	165	161	499
6	160	155	660	6	143	139	453
7	161	160	242	7	131	126	482
8	169	166	276	8	128	120	500
9	169	169	969	9	142	+131	437
10	161	160	538	10	144	136	542
avg.	168.6	166.1	715.2	11	121	117	412
				12	137	130	355
				13	171	163	1165
				14	146	142	530
				15	152	145	456
				avg.	139.4	133.5	503.0

instance sets (see subfigures (a) to (d) of Figure 2). The left graphic of each subfigure has to be read as follows. The  $x$ -axis ranges over the possible values for  $l$ , while the  $y$ -axis indicates the size of the set of common blocks that is used for solving models  $ILP_{ph1}$  and  $ILP_{ph2}$ . The graphic shows two curves. The one with a black line concerns solving model  $ILP_{ph1}$  in phase 1 of the heuristic, while the other one (shown by means of a grey line) concerns solving model  $ILP_{ph2}$  in phase two of the heuristic. The dots indicate for each value of  $l$  the size of the set of common blocks used by the corresponding models. Moreover, in case the interior of a dot is light-grey (yellow in the online version) this means that the corresponding model could not be solved to optimality within 50 CPU seconds, while a black interior of a dot indicates that the corresponding model was solved to optimality. Finally, the bars in the background of the graphic present the values of the solutions that were generated with different values of  $l$ . The graphics on the right hand side present the corresponding computation times required by solving the different models.

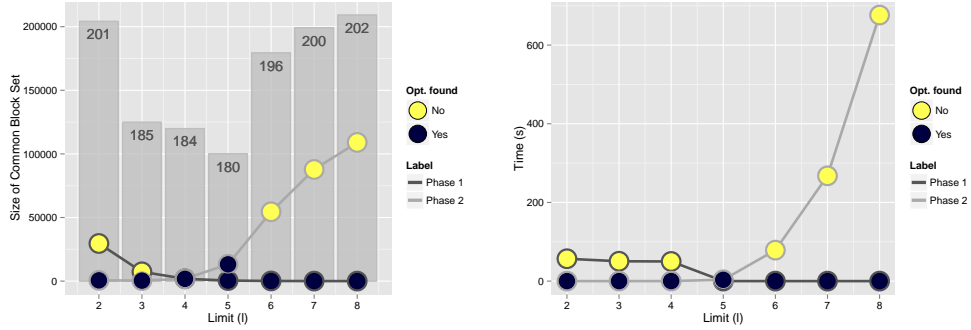
The following observations can be made. When the value of  $l$  is close to the lower or



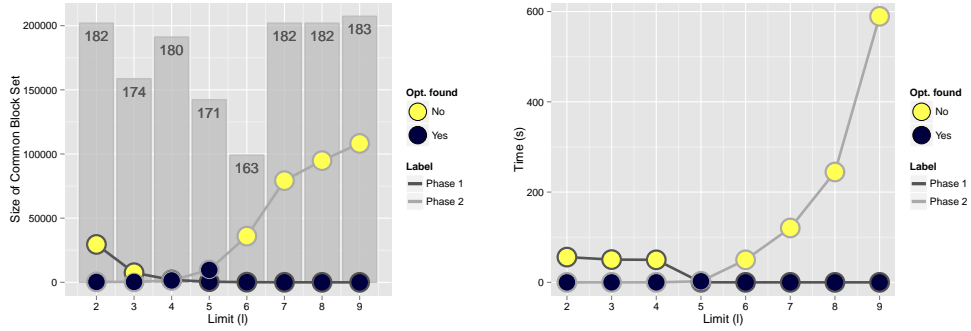
(a) Instance 10 of GROUP1, results (left) and computation time (right)



(b) Instance 5 of GROUP2, results (left) and computation time (right)



(c) Instance 3 of GROUP3, results (left) and computation time (right)



(d) Instance 13 of REAL, results (left) and computation time (right)

Figure 2: Detailed information on the results of the proposed heuristic for four chosen problem instances. The description of the information content of the graphics is provided in the text.

Table 7: Results of applying the heuristic in the context of larger instances.

length	CPLEX	Heuristic				
	value	value	time (s)	$ B_{\geq 5} $	$ B_{ph2} $	% of $B$
800	210	225	16	801	13138	6.5%
1000	304	279	54	1385	12714	4.2%
1200	342	326	70	1785	27852	6.2%
1400	401	378	61	2535	20628	3.5%
1600	442	413	65	3244	24708	3.3%
1800	486	473	99	4416	44139	4.5%
2000	n.a.	518	126	5132	61731	5.0%

the upper bound—that is, either close to 2 or close to  $l_{\max}$ —one of the two involved sets of common blocks is quite large, and, therefore, the computation time needed for solving the corresponding ILP may be large, in particular when the input instance is rather large. On the contrary, for intermediate values of  $l$ , the size of both involved sets of common blocks is moderate, and, therefore, CPLEX is rather fast in providing solutions, even if the optimal solution is not found (or can not be proven) within 50 CPU seconds. Moreover, the best results are usually obtained for intermediate values of  $l$ . This is with the exception of instance 10 of GROUP1, which might be an anomaly caused by the rather small size of the problem instance.

### 3.4 Results of Heuristic for Larger Instances

Based on the findings of the previous subsection the heuristic was applied with an intermediate value of  $l = 5$  to all problem instances from the set of larger instances described at the end of Section 2.4.2. The results are shown in Table 7. The first table column provides the length of the input strings of the corresponding random instance. The second column indicates the result of applying CPLEX with a computation time limit of 3600 CPU seconds to  $ILP_{\text{orig}}$ .<sup>6</sup> The remaining five columns contain the results of heuristic. The first one of these columns provides the value of the solution generated by the heuristic, while the second column shows the corresponding computation time. The next two columns provide the size of the sets of common blocks used in phase 1, respectively phase 2, of the heuristic. Finally, the last column gives information about the number of common blocks considered by the heuristic in comparison to the size of the complete set of common blocks (which can be found in Table 5). In particular, summing the common block set sizes from phases 1 and 2 of the heuristic and comparing this number with the size of the complete set of common blocks, the percentage of the common blocks considered by the heuristic can easily be calculated. This percentage is given in the last table column. As in all tables of this paper, the best result per table row is marked by a grey background.

The following observations can be made. First, apart from the smallest problem instance, the heuristic outperforms the application of CPLEX to model  $ILP_{\text{orig}}$ . Moreover, this is achieved in a fraction of the time needed by CPLEX. Finally, it is reasonable to assume that the success of the heuristic is due to an important reduction of the common blocks that are considered (see last table column). In general, the heuristic only considers between 3.3% and 6.5% of all common blocks. This is why the computation times are rather low in comparison to CPLEX.

<sup>6</sup>Remember that the results of applying CPLEX to  $ILP_{\text{orig}}$  were described in detail in Section 2.4.2.

## 4 Conclusions and Future Work

In this paper we considered a problem with applications in bioinformatics known as the minimum common string partition problem. First, we introduced an integer linear programming model for this problem. By applying the IBM ILOG CPLEX solver to this model we were able to improve all best-known solutions from the literature for a problem instance set consisting of 45 instances of different sizes. The smallest ones of these problem instances could even be solved to optimality in very short computation time. The second contribution of the paper concerned a 2-phase heuristic which is strongly based on the developed integer linear programming model. The results have shown that, first, the heuristic outperforms competitor algorithms from the literature, and second, that it is applicable to larger problem instances.

Concerning future work, we aim at studying the incorporation of mathematical programming strategies based on the introduced integer linear programming model into metaheuristic techniques such as GRASP and iterated greedy algorithms. Moreover, we aim at identifying other string-based optimization problems for which a 2-phase strategy such as the one introduced in this paper might work well.

**Acknowledgements** C. Blum was supported by project TIN2012-37930 of the Spanish Government. In addition, support is acknowledged from IKERBASQUE (Basque Foundation for Science). J. A. Lozano was partially supported by the Saiotek and IT609-13 programs (Basque Government), TIN2010-14931 (Spanish Ministry of Science and Innovation), COM-BIOMED network in computational bio-medicine (Carlos III Health Institute)

## References

- [1] C. Blum, J. A. Lozano, and P. Pinacho Davidson. Iterative probabilistic tree search for the minimum common string partition problem. In M. J. Blesa, C. Blum, and S. Voss, editors, *Proceedings of HM 20104– 9th International Workshop on Hybrid Metaheuristics*, Lecture Notes in Computer Science. Springer Verlag, Berlin, Germany, 2014. In press.
- [2] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Computing the assignment of orthologous genes via genome rearrangement. In *Proceedings of the Asia Pacific Bioinformatics Conference 2005*, pages 363–378, 2005.
- [3] M. Chrobak, P. Kolman, and J. Sgall. The greedy algorithm for the minimum common string partition problem. In K. Jansen, S. Khanna, J. D. P. Rolim, and D. Ron, editors, *Proceedings of APPROX 2004 – 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, volume 3122 of *Lecture Notes in Computer Science*, pages 84–95. Springer Berlin Heidelberg, 2004.
- [4] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Transactions on Algorithms*, 3(2):1–19, 2007.
- [5] P. Damaschke. Minimum common string partition parameterized. In K. A. Crandall and J. Lagergren, editors, *Proceedings of WABI 2008 – 8th International Workshop on Algorithms in Bioinformatics*, volume 5251 of *Lecture Notes in Computer Science*, pages 87–98. Springer Berlin Heidelberg, 2008.

- [6] S. M. Ferdous and M. S. Rahman. Solving the minimum common string partition problem with the help of ants. In Y. Tan, Y. Shi, and H. Mo, editors, *Proceedings of ICSI 2013 – 4th International Conference on Advances in Swarm Intelligence*, volume 7928 of *Lecture Notes in Computer Science*, pages 306–313. Springer Berlin Heidelberg, 2013.
- [7] S. M. Ferdous and M. S. Rahman. A MAX-MIN ant colony system for minimum common string partition problem. *CoRR*, abs/1401.4539, 2014. <http://arxiv.org/abs/1401.4539>.
- [8] B. Fu, H. Jiang, B. Yang, and B. Zhu. Exponential and polynomial time algorithms for the minimum common string partition problem. In W. Wang, X. Zhu, and D.-Z. Du, editors, *Proceedings of COCOA 2011 – 5th International Conference on Combinatorial Optimization and Applications*, volume 6831 of *Lecture Notes in Computer Science*, pages 299–310. Springer Berlin Heidelberg, 2011.
- [9] M. R. Garey and D. S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness*. W. H. Freeman, 1979.
- [10] A. Goldstein, P. Kolman, and J. Zheng. Minimum common string partition problem: Hardness and approximations. In R. Fleischer and G. Trippen, editors, *Proceedings of ISAAC 2004 – 15th International Symposium on Algorithms and Computation*, volume 3341 of *Lecture Notes in Computer Science*, pages 484–495. Springer Berlin Heidelberg, 2005.
- [11] I. Goldstein and M. Lewenstein. Quick greedy computation for minimum common string partitions. In R. Giancarlo and G. Manzini, editors, *Proceedings of CPM 2011 – 22nd Annual Symposium on Combinatorial Pattern Matching*, volume 6661 of *Lecture Notes in Computer Science*, pages 273–284. Springer Berlin Heidelberg, 2011.
- [12] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Computer Science and Computational Biology. Cambridge University Press, Cambridge, 1997.
- [13] D. He. A novel greedy algorithm for the minimum common string partition problem. In I. Mandoiu and A. Zelikovsky, editors, *Proceedings of ISBRA 2007 – Third International Symposium on Bioinformatics Research and Applications*, volume 4463 of *Lecture Notes in Computer Science*, pages 441–452. Springer Berlin Heidelberg, 2007.
- [14] W. J. Hsu and M. W. Du. Computing a longest common subsequence for a set of strings. *BIT Numerical Mathematics*, 24(1):45–59, 1984.
- [15] H. Jiang, B. Zhu, D. Zhu, and H. Zhu. Minimum common string partition revisited. *Journal of Combinatorial Optimization*, 23(4):519–527, 2012.
- [16] H. Kaplan and N. Shafrir. The greedy algorithm for edit distance with moves. *Information Processing Letters*, 97(1):23–27, 2006.
- [17] P. Kolman. Approximating reversal distance for strings with bounded number of duplicates. In J. Jędrzejowicz and A. Szepietowski, editors, *Proceedings of MFCS 2005 – 30th International Symposium on Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 580–590. Springer Berlin Heidelberg, 2005.



- [18] P. Kolman and T. Waleń. Reversal distance for strings with duplicates: Linear time approximation using hitting set. In T. Erlebach and C. Kaklamanis, editors, *Proceedings of WAOA 2007 – 4th International Workshop on Approximation and Online Algorithms*, volume 4368 of *Lecture Notes in Computer Science*, pages 279–289. Springer Berlin Heidelberg, 2007.
- [19] C.N. Meneses, C.A.S. Oliveira, and P.M. Pardalos. Optimization techniques for string selection and comparison problems in genomics. *IEEE Engineering in Medicine and Biology Magazine*, 24(3):81–87, 2005.
- [20] S.R. Mousavi, M. Babaie, and M. Montazerian. An improved heuristic for the far from most strings problem. *Journal of Heuristics*, 18:239–262, 2012.
- [21] D. Shapira and J. A. Storer. Edit distance with move operations. In A. Apostolico and M. Takeda, editors, *Proceedings of CPM 2002 – 13th Annual Symposium on Combinatorial Pattern Matching*, volume 2373 of *Lecture Notes in Computer Science*, pages 85–98. Springer Berlin Heidelberg, 2002.
- [22] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.